Depth Efficient Neural Networks for Division and Related Problems

Kai-Yeung Siu, *Member, IEEE,* Jehoshua Bruck, *Member, IEEE,* Thomas Kailath, *Fellow, IEEE,* and Thomas Hofmeister

Abstract— **An artificial neural network (ANN) is commonly modeled by a threshold circuit, a network of interconnected processing units called linear threshold gates. The depth of a circuit represents the number of unit delays or the time for parallel computation. The size of a circuit is the number of gates and measures the amount of hardware. It was known that traditional logic circuits consisting of only unbounded fan-in AND, OR, NOT gates would require at least $\Omega(\log n/\log\log n)$ depth to compute common arithmetic functions such as the product or the quotient of two n-bit numbers, if the circuit size is polynomially bounded (in n). It is shown that ANN's can be much more powerful than traditional logic circuits, assuming that each threshold gate can be built with a cost that is comparable to that of AND/OR logic gates. In particular, the main results show that powering and division can be computed by polynomial-size ANN's of depth 4, and multiple product can be computed by polynomial-size ANN's of depth 5. Moreover, using the techniques developed here, a previous result can be improved by showing that the sorting of n n-bit numbers can be carried out in a depth-3 polynomial size ANN. Furthermore, it is shown that the sorting network is optimal in depth.**

Index Terms—**Neural networks, threshold circuits, circuit complexity, division, arithmetic functions.**

I. Introduction

RECENT interest in the application of artificial neural networks [18], [20] has spurred research interest in the theoretical study of such networks. In most models of neural networks, the basic processing unit is a Boolean gate that computes a linear threshold function, or an analog element that computes a sigmoidal function. Artificial neural networks can be viewed as circuits of these processing units which are massively interconnected together. After Hopfield [18] introduced his neural network model of associative memory, many researchers have tried to analyze the "capacity" or "convergence" of the discrete model and many other models of associative memory have been proposed. Not long after, Rumelhart [17] and his group of researchers proposed a model of feedforward multilayer networks of analog sigmoidal functional elements and invented the "back-propagation" heuristic algorithm for "training" (i.e., adaptively changing) the parameters in the network to perform a desired function. Many researchers have since then applied and modified the algorithm to obtain excellent empirical results. In some applications, artificial neural networks seem to outperform the traditional methods [4].

While neural networks have found wide application in many areas, the behavior and the limitation of these networks are far from being understood. One common model of a neural network is a *threshold circuit*. Incidentally, the study of threshold circuits, motivated by some other complexity theoretic issues, has also gained much interest in the area of computer science. Threshold circuits are Boolean circuits in which each gate computes a linear threshold function, whereas in the classical model of *unbounded fan-in* Boolean circuits only AND, OR, NOT gates are allowed. A Boolean circuit is usually arranged in layers such that all gates in the same layer are computed concurrently and the circuit is computed layer by layer in some increasing *depth* order. We define the *depth* as the number of layers in the circuit. Thus, each layer represents a unit delay and the depth represents the overall delay in the computation of the circuit.

A. Related Work

Theoretical computer scientists have used *unbounded fan-in* Boolean circuits as a model to understand fundamental issues of parallel computation. To be more specific, this computation model should be referred to as *unbounded fan-in* parallelism, since the number of inputs to each gate in the Boolean circuit is not bounded by a constant. The theoretical study of unbounded fan-in parallelism may give us insights into devising faster algorithms for various computational problems than would be possible with bounded fan-in parallelism. In fact, any nondegenerate Boolean function of n variables requires at least $\Omega(\log n)$ depth to compute in a bounded fan-in circuit. On the other hand, in some practical situations (for example, large fan-in circuits such as programmable logic arrays (PLA's) or multiple processors simultaneously accessing a shared bus), unbounded fan-in parallelism seems to be a natural model. For example, a PLA can be considered as a depth-2 AND/OR circuit.

In the Boolean circuit model, the amount of resources is usually measured by the number of gates, and is considered

Manuscript received January 31, 1992; revised July 29, 1992. This work was supported in part by an Irvine Faculty Research Grant 91/92-27 and by the Joint Services Program at Stanford University (U.S. Army, U.S. Navy, U.S. Air Force) under Contract DAAL03-88-C-0011, and the Department of the Navy (NAVELEX) under Contract N00039-84-C-0211, NASA Headquarters, Center for Aeronautics, and Space Information Sciences under Grant NAGW-419-S6. The work of T. Hofmeister was supported by DFG Grant We 1066/2-2.

K.-Y. Siu is with the Department of Electrical and Computer Engineering, University of California, Irvine, CA 92717.

J. Bruck is with the IBM Research Division, Almaden Research Center, K54/802, 650 Harry Road, San Jose, CA 95120-6099.

T. Kailath is with the Information Systems Laboratory, Stanford University, Stanford, CA 94305.

T. Hofmeister is with the Lehrstuhl Informatik II, Universität Dortmund, Postfach 50 05 00, 4600 Dortmund 50, Germany.

IEEE Log Number 9207218.

to be "reasonable" as long as it is bounded by a polynomial (as opposed to exponential) in the number of the inputs. For example, a Boolean circuit for computing the sum of two n-bit numbers with $O(n^3)$ gates is "reasonable," although circuit designers might consider the size of the circuit impractical for moderately large n. One of the most important theoretical issues in parallel computation is the following: *given that the number of gates in the Boolean circuit is bounded by a polynomial in the size of inputs, what is the minimum depth* (i.e., *number of layers*) *that is needed to compute certain functions?*

A first step toward answering this important question was taken by Furst *et al.* [11] and independently by Ajtai [2]. It follows from their results that for many basic functions, such as the *parity* and the *majority* of n Boolean variables, or the multiplication of two n-bit numbers, any constant depth (i.e., independent of n) classical Boolean circuit of unbounded fan-in AND/OR gates computing these functions must have more than a polynomial (in n) number of gates. This lower bound on the size was subsequently improved by Yao [30] and Håstad [13]; it was proved that indeed an exponential number of AND/OR gates are needed. So functions such as *parity* or *majority* are computationally "hard" with respect to constant depth and polynomial size classical Boolean circuits.[1] Another way of interpreting these results is that circuits of AND/OR gates computing these "hard" functions which use a polynomial amount of chip area must have *unbounded delay* (i.e., delay that increases with n). In fact, the lower bound results imply that the minimum possible delay for multipliers (with polynomial number of AND/OR gates) is $\Omega(\log n/\log\log n)$. These results also give theoretical justification why it is impossible for circuit designers to implement fast parity circuits or multipliers in small chip area using AND, OR gates as the basic building blocks.

One of the "hard" functions previously mentioned is the *majority* function, a special case of a threshold function in which the *weights* or parameters are restricted. A natural extension is to study Boolean circuits that contain *majority gates*. This type of Boolean circuit is called a threshold circuit and is believed to capture some aspects of the computation in our brain [21]. In the rest of the paper, the term "neural networks" refers to the threshold circuits model.

With the addition of majority gates, the resulting Boolean circuit model seems much more powerful than the classical one. Indeed, it was first shown by Muroga [22] three decades ago that any symmetric Boolean function (e.g., parity) can be computed by a two-layer neural network with $(n+1)$ gates. Recently, Chandra *et al.* [10] showed that multiplication of two n-bit numbers and sorting of n n-bit numbers can be computed by neural networks with "constant" depth and polynomial size. These "constants" have been significantly reduced by Siu and Bruck [28], [29] to 4 in both cases, whereas a lower bound of depth 3 was proved by Hajnal *et al.* [12] in the case of multiplication. A depth-4 neural network for multiplication was independently discovered in [16] with a smaller size $O(n^2)$ than the result in [28]. On the other hand, the existence of small constant-depth and polynomial-size neural networks for division was still left as an open problem. In [17], it was shown that any polynomial-size threshold circuit with polynomially bounded integer weights computing division must have depth at least 3.

While it was known that multiplication of two n-bit numbers can be computed by an $O(\log n)$ depth classical Boolean circuit [31], it had been a challenge to complexity theorists to construct polynomial-size $O(\log n)$-depth circuits that compute division. The question whether division has the same *parallel circuit complexity* as multiplication in the classical Boolean circuit model was finally resolved by Beame *et al.* [6]. Using a well-known result from number theory, they were able to construct polynomial-size $O(\log n)$-depth classical Boolean circuits that compute division. It was later shown by Reif and Tate [26], [27] that their construction can be modified to obtain "constant" depth neural networks for division and related problems. However, the issue of how small the "constant" can be was not examined in [26]. In fact, it was not clear from the construction of Beame *et al.* whether the "constant" can be reduced to below 20. Because division is one of the most fundamental operations in arithmetic computations, for both practical and theoretical reasons, it is interesting to determine the smallest possible depth required for division circuits.

Our main contribution in this paper is to show that small constant depth neural networks for division and related problems can be constructed. These results have the following implication on their practical significance: *suppose we can use analog devices to build threshold gates with a cost (in terms of delay and chip area) that is comparable to that of AND, OR logic gates, then we can compute many basic functions much faster than using traditional circuits*. Clearly, the particular weighting of depth, fan-in, and size that gives a realistic measure of a network's cost and speed depends on the technology used to build it. One case where circuit depth would seem to be the most important parameter is when the circuit is implemented using optical devices. We refer those who are interested in the optical implementation of neural networks to [1].

Our techniques are adapted from those in [28] and [6]. We shall follow the complexity theoretic approach mentioned above and focus our study on the depth of neural networks; we allow ourselves to neglect issues on the size so long as the size is bounded by a polynomial in the number of inputs. The question whether the size and the depth can both be simultaneously reduced to the smallest possible is left for future research. We would also like to mention that the circuits constructed in this paper are not *uniform*. (For those who are interested in various notions of uniformity of a circuit family, see [32].)

B. Outline of the Paper

The rest of this paper is divided into three major sections. In Section II, we shall give a formal definition of threshold circuits, our model for neural networks, and introduce related concepts. We then present our main results in Section III. This section is further divided into several subsections. Since techniques similar to the construction in [28] of depth-4 neural networks for multiplication of two n-bit numbers will be

[1] An excellent survey of results on the complexity of Boolean functions can be found in [7] and [32].

used later on, we shall review these results and present the underlying ideas first. By generalizing the idea of computing symmetric functions, we show how to carry out the exponentiation and modulo a "small" integer in depth-2 neural networks. Then combining these techniques and the results in [6] and [16], we show that powering can be computed in depth-4 neural networks and multiple product can be computed in depth-5 neural networks. Based on this construction, we show that division can be computed in a depth-4 neural network. Moreover, we show that similar techniques can be applied to obtain a depth-3 neural network for sorting, which is an improvement of our earlier result in [29]. We also show that our sorting network is optimal in depth. Finally, in Section IV, we conclude with some open problems and future directions for research.

II. Preliminaries

The purpose of this section is to introduce terminologies and the necessary background for understanding the main results in this paper.

Definition 1: A *linear threshold function* $f(X)$: $\{0,1\}^n \rightarrow \{0, 1\}$ is a Boolean function such that

$$f(X) = \operatorname{sgn}(F(X)) = \begin{cases} 1, & \text{if } F(X) \geq 0, \\ 0, & \text{if } F(X) < 0, \end{cases}$$

where

$$F(X) = \sum_{i=1}^{n} w_i \cdot x_i + w_0$$

$$X = (x_1, \cdots, x_n) \in \{0, 1\}^n.$$

The real value coefficients w_i are commonly referred to as the *weights* of the threshold function. A *linear threshold gate* is a Boolean gate that computes a linear threshold function.

Remark 1: Although the definition of a linear threshold function allows the weights to be real numbers, it is known [23] that we can replace each of the weights by integers of $O(n \log n)$ bits, where n is the number of input Boolean variables. So, in the rest of the paper, we shall assume without loss of generality that all weights are integers. In fact, we shall only study a subclass of linear threshold functions in which each function $f(X) = \operatorname{sgn}(\sum_{i=1}^{n} w_i \cdot x_i + w_0)$ is characterized by the property that the weights w_i are integers and bounded by a polynomial in n, i.e., $|w_i| \leq n^c$ for some constant $c > 0$. Furthermore, it is well known that the w_i's can be chosen so that $\sum_{i=1}^{n} w_i \cdot x_i + w_0 \neq 0$ for all inputs $(x_1, \cdots, x_n) \in \{0, 1\}^n$. To see this, simply note that if $F(X)$ takes on integer values for all X, then $\operatorname{sgn}\{F(X)\} = \operatorname{sgn}\{2F(X) + 1\}$, and $2F(X) + 1 \neq 0$ for all X. *Hereafter, we assume that all linear threshold gates in our threshold circuits have these properties.*

Definition 2: A *threshold circuit* [14], [25] is a Boolean circuit of linear threshold gates. The *size* of a threshold circuit is the number of gates in the circuit. The *depth* of a gate is the maximum number of gates along any directed path from the inputs to that gate. The *depth* of the circuit is the maximum depth of all output gates. If we group all gates with the same depth together, we can consider the circuit to be arranged in layers, where the depth of the circuit is equal to the number of layers (excluding the input layer) in the circuit, and gates of the same layer are computed concurrently. Thus, the depth of the circuit can be interpreted as the time for parallel computation.

As mentioned in Section I, we shall use threshold circuits as a model to study the computation in artificial neural networks. So we shall use the terms "threshold circuits" and "neural networks" interchangeably. *Moreover, all our results will be stated with the understanding that the sizes of the networks are all polynomially bounded.* Note that the above definition does not impose any restriction on the *fan-in* and *fan-out* of each threshold gate in the network, i.e., each gate can have arbitrarily many inputs from a previous layer, and can feed its output to arbitrarily many gates in subsequent layers. Since neural networks are characterized by their massive parallelism, the unbounded fan-in parallelism of threshold circuits seems a natural assumption. Moreover, threshold circuits with bounded fan-in have *exactly* the same power as traditional Boolean circuits with bounded fan-in (see [32]).

Suppose we want to compute the quotient of two integers. Since some quotient in binary representation might require infinitely many bits, a Boolean circuit can only compute the most significant bits of the quotient. If a number has both finite and infinite binary representation (for example $0.1 = 0.0111\cdots$), we shall always express the number in its finite binary representation. One of our main results in this paper is to show that the following division problem can be computed in small depth neural networks.

Definition 3: Let X and $Y \geq 1$ be two input n bit integers. Let $X/Y = \sum_{i=-\infty}^{n-1} z_i 2^i$ be the quotient of X divided by Y. We define $\operatorname{DIV}_k(X/Y)$ to be X/Y *truncated* to the nearest $(n + k)$-bit number, i.e.,

$$\operatorname{DIV}_k(X/Y) = \sum_{i=-k}^{n-1} z_i 2^i.$$

In particular, $\operatorname{DIV}_0(X/Y)$ is $\lfloor X/Y \rfloor$, the greatest integer $\leq X/Y$.

The above problem is to find the quotient of two integers with truncation, which is a commonly adopted definition of the division problem. Similarly, the "rounded" quotient can also be computed by depth-4 polynomial-size neural networks using the techniques developed in this paper.

We shall see that the following function POWERING is closely related to the division problem.

Definition 4: Let X be an input n-bit number. We define POWERING to be the n^2-bit representation of X^n.

A function similar in nature to POWERING is EXPONENTIATION.

Definition 5: Let Y be an input $o(\log n)$-bit number (i.e., $Y \leq n^k$ for some constant k), and $c \geq 0$ be a fixed integer. We define EXPONENTIATION to be the $O(n^k)$-bit representation of c^Y.

Remark 2: Since we are only concerned with functions that can be computed by polynomial size circuits, it is necessary that the outputs of the function be representable by polynomial number of bits. The assumption in the function EXPONEN-

TIATION that the exponent Y is polynomially bounded is simply to guarantee that c^Y can be represented in a polynomial number of bits. It turns out that because of this restriction, EXPONENTIATION is easier to compute than POWERING.

We shall need some number theoretic terminologies in presenting our main results. (For an introduction to number theory, see [24].)

Definition 6: Let x, y, m be positive integers. We write $x \equiv y \pmod m$ if $(x-y)$ is divisible by m, and $x \bmod m$ to denote the unique integer z such that $0 \le z < m$ and $x \equiv z \pmod m$.

We shall make use of the following results in number theory.

Fermat's Theorem: Let p be a prime number. Then for any positive integer a not divisible by p, $a^{p-1} \equiv 1 \bmod p$.

Chinese Remainder Theorem: Let p_i for $i = 1, \cdots, n$ be relatively prime numbers, and $P_n = \prod_{i=1}^{n} p_i$. Let $0 \le Z < P_n$, $q_i = P_n/p_i$, $\tilde{q}_i$ be an integer x such that $1 = q_i x \bmod p_i$, and $m_i = q_i \cdot \tilde{q}_i$ (Note that $\tilde{q}_i$ exists since q_i and p_i are relatively prime.) Further, let $r_i \equiv Z \pmod{p_i}$. Then

$$Z = \left(\sum_{i=1}^{n} r_i \cdot m_i \right) \bmod P_n.$$

Example: Let $Z = 14$, $p_1 = 3$, $p_2 = 5$. Then $P_n = 15$, $q_1 = 5$, $q_2 = 3$, $\tilde{q}_1 = 2$, $\tilde{q} = 2$, $m_1 = 10$, $m_2 = 6$, $r_1 = 2$, $r_2 = 4$. So $Z = (r_1 m_1 + r_2 m_2) \bmod P_n = 44 \bmod 15 = 14$.

The Chinese Remainder Theorem enables us to represent a number in a "mixed radix" system (as opposed to the traditional "fixed radix" system), where the r_i is the "coefficient" with respect to the "radix" m_i. This representation is useful because sometimes it is much easier to compute the coefficients r_i's than Z itself.

Notation: In this paper, we will be mainly interested in *asymptotic results*. So it will be convenient to use some standard shorthand notations. For nonnegative functions f and g, when we write $f(n) = O(g(n))$ or $g(n) = \Omega(f(n))$, we shall mean that $f(n) \le c \cdot g(n)$ for some constant $c > 0$, as $n \to \infty$.

Since we shall be interested only in neural networks of polynomial size, for convenience, we shall say that $f(X)$ can be computed in depth-d neural networks instead of saying that $f(X)$ can be computed in depth-d polynomial-size neural networks.

III. Main Results

A. Multiplication in Depth-4

One key idea in constructing small depth neural networks for division and related problems is that we can compute the sum of many (exponentially large) numbers in a depth-3 network. This result leads to the construction of a depth-4 neural network for multiplication in [28] and [16]. Since this idea is crucial in understanding our main results, we first review the techniques here.

The "Block Save" Technique: The underlying idea of computing the sum of many large numbers is to reduce this multiple sum to the sum of only two numbers. This technique, which we call the "block save" technique, is a generalization of the traditional "carry save" technique. The main difficulty in computing the sum of many numbers is to compute "carry" bits in parallel. The traditional "carry save" technique reduces the sum of three numbers to the sum of two numbers in one step, where one of the resulting two numbers only consists of the carry bits. Let the original three numbers be $X = x_{n-1}x_{n-2}\cdots x_0$, $Y = y_{n-1}y_{n-2}\cdots y_0$, $Z = z_{n-1}z_{n-2}\cdots z_0$ (all numbers are in binary representation). The ith bit of each number can be added to give $x_i + y_i + z_i = 2c_{i+1} + w_i$, where the c_{i+1} is the carry bit generated by the sum of the ith bits x_i, y_i, and z_i. Thus, $X + Y + Z = C + W$, where $C = c_n c_{n-1}\cdots c_0$ with $c_0 = 0$ and $W = w_{n-1}w_{n-2}\cdots w_0$. Note that $c_{i+1} = x_i y_i \vee y_i z_i \vee z_i x_i$ and $w_i = x_i \oplus y_i \oplus z_i$, hence, the c_{i+1}'s and w_i's can all be computed in parallel in one step. For example, let

$$X = 1001, \quad Y = 0111, \quad Z = 1101,$$
$$\text{then } W = 0011 \quad \text{and} \quad C = 11010.$$

The "block save" technique generalizes this idea and reduces the sum of n $O(n)$-bit numbers to the sum of two $O(n)$-bit numbers in one step. We now give a sketch of this "block save" idea for the case when each of the n numbers has exactly n bits. (For details of the proof, see [28].)

Denote each of the n-bit numbers by $x_i = x_{i_{n-1}}x_{i_{n-2}}\cdots x_{i_0}$, for $i = 1, \cdots n$. For simplicity, assume $\log n$ and $N = n/\log n$ to be an integer. Partition each binary number x_i into N consecutive blocks $\tilde{x}_{i_0}, \tilde{x}_{i_1}, \cdots, \tilde{x}_{i_{N-1}}$ of $(\log n)$-bits each so that

$$x_i = \sum_{j=0}^{N-1} \tilde{x}_{i_j} \cdot 2^{\log n \cdot j},$$

where $0 \le \tilde{x}_{i_j} < 2^{\log n}$. Hence, the total sum (after rearranging the indices of summation) becomes

$$S = \sum_{i=1}^{n} x_i = \sum_{j=0}^{N-1} \left(\sum_{i=1}^{n} \tilde{x}_{i_j} \right) \cdot 2^{\log n \cdot j}.$$

Observe that for each $j = 0, \cdots, N-1$, the block sum

$$\tilde{s}_j = \sum_{i=1}^{n} \tilde{x}_{i_j} < \sum_{i=1}^{n} 2^{\log n} = 2^{2 \log n}$$

and thus $\tilde{s}_j$ can be represented in $2 \log n$ bits. Therefore, each block sum $\tilde{s}_j$ can be expressed as

$$\tilde{s}_j = \tilde{c}_{j+1} 2^{\log n} + \tilde{w}_j$$

where $0 \le \tilde{c}_{j+1} < 2^{\log n}$ is the $(\log n)$-bit carry of the sum $\tilde{s}_j$ with $\tilde{c}_0 = 0$, and $\tilde{w}_j < 2^{\log n}$. Thus,

$$S = \sum_{j=0}^{N-1} \left(\sum_{i=1}^{n} \tilde{x}_{i_j} \right) \cdot 2^{\log n \cdot j} = \sum_{j=0}^{N-1} \tilde{s}_j 2^{\log n \cdot j}$$
$$= \sum_{j=0}^{N-1} \tilde{c}_{j+1} 2^{(j+1) \log n} + \sum_{j=0}^{N-1} \tilde{w}_j 2^{j \log n}.$$

Since both $\tilde{c}_{j+1}$ and $\tilde{w}_j$ are $< 2^{\log n}$, the binary representation of $\sum_{j=0}^{N-1} \tilde{c}_{j+1} 2^{(j+1)\log}$ is simply the concatenation of the bits $\tilde{c}_N \tilde{c}_{N-1} \cdots \tilde{c}_0$. Similarly, concatenating the bits $\tilde{w}_{N-1} \cdots \tilde{w}_0$ gives the binary representation of $\sum_{j=0}^{N-1} \tilde{w}_j 2^{j \log n}$. Hence, we have shown the reduction from the sum of n n-bit numbers to the sum of two $O(n)$-bit numbers. Furthermore, we can compute all $\tilde{c}_j$'s and $\tilde{w}_j$'s in parallel to obtain the resulting two numbers.

It remains to show how to compute the $(2 \log n)$-bit representation of each block sum $\tilde{s}_j = \tilde{c}_{j+1} 2^{\log n} + \tilde{w}_j$. In fact, we shall see that each block sum can be computed with two layers of threshold gates. This result is implied by the following lemma (see also Theorem 1 in [5] for a more general technique).

Lemma 1: Let $X = (x_{n-1}, x_{n-2}, \cdots, x_0) \in \{0, 1\}^n$, and $f: \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function of X which depends only on a polynomially bounded weighted sum of the input variables. Then $f(X)$ can be represented as a sum of polynomially many linear threshold functions in X. Hence, $f(X)$ can be computed in a depth-2 neural network.

Proof: By assumption, we can write $f(X) = \tilde{f}(\sum_{i=0}^{n-1} w_i x_i)$ where w_i are integers bounded by a polynomial in n. Let $N = \sum_{i=0}^{n-1} |w_i|$. There exists a set of s disjoint subintervals in $[-N, -N]$, say, $[k_1, \tilde{k}_1], [k_2, \tilde{k}_2], \cdots, [k_s, \tilde{k}_s]$ (k_j's, $\tilde{k}_j$'s are integers, and possibly $k_j = \tilde{k}_j$) such that

$$f(X) = 1 \text{ iff } \sum_{i=0}^{n-1} w_i x_i \in [k_j, \tilde{k}_j], \qquad \text{for some } j.$$

Let

$$y_{k_j} = \operatorname{sgn}\left\{\sum_{i=0}^{n-1} w_i x_i - k_j\right\}; \quad \tilde{y}_{k_j} = \operatorname{sgn}\left\{\tilde{k}_j - \sum_{i=0}^{n-1} w_i x_i\right\} \qquad \text{for } j = 1, \cdots, s.$$

We claim

$$f(X) = \sum_{j=1}^{s} (y_{k_j} + \tilde{y}_{k_j}) - s.$$

Note the following.

If for all j,

$$\sum_{i=0}^{n-1} w_i x_i \notin [k_j, \tilde{k}_j],$$

then

$$y_{k_j} + \tilde{y}_{k_j} = 1, \qquad \text{for all } j = 1, \cdots, s.$$

Thus,

$$\sum_{j=1}^{s} (y_{k_j} + \tilde{y}_{k_j}) - s = 0.$$

On the other hand, if

$$\sum_{i=0}^{n-1} w_i x_i \in [k_j, \tilde{k}_j] \qquad \text{for some } j \in \{1, \cdots, s\}$$

then

$$y_{k_j} + \tilde{y}_{k_j} = 2 \qquad \text{and} \qquad y_{k_i} + \tilde{y}_{k_i} = 1, \qquad \text{for} \quad i \neq j.$$

Thus,

$$\sum_{i=1}^{s} (y_{k_i} + \tilde{y}_{k_i}) - s = s + 1 - s = 1.$$

Now, s is at most $N = \sum_{i=0}^{n-1} |w_i|$, which is bounded by a polynomial in n. Hence, there can be at most polynomially many y_{k_j}'s and $\tilde{y}_{k_j}$'s.

To compute $f(X)$ with a depth-2 threshold circuit, the first layer of our circuit consists of threshold gates which compute the values y_{k_j} and $\tilde{y}_{k_j}$. In the second layer, the output gate takes as inputs y_{k_j}, $\tilde{y}_{k_j}$ and outputs $\operatorname{sgn}\{2[\sum_{j=1}^{s}(y_{k_j} + \tilde{y}_{k_j}) - s] - 1\}$. □

Recall that in our "block save" technique, a crucial step is to compute the $(2 \log n)$-bit representation of each "block sum" $\tilde{s}_j = \tilde{c}_{j+1} 2^{\log n} + \tilde{w}_j$ (a sum of n $(\log n)$-bit numbers). Since each number of the summands is of the form $\sum_{k=0}^{\log n - 1} 2^k x_k < n$, the total sum is still a polynomially bounded weighted sum of the $n \log n$ variables x_{i_l}'s, where $1 \leq i \leq n$ and $j \log n \leq l < (j+1) \log n$. Thus, each bit of $\tilde{c}_{j+1}$ and $\tilde{w}_j$ is a function of a polynomially bounded weighted sum of $n \log n$ variables. By the above lemma, each block sum $\tilde{s}_j$ can be computed in a depth-2 neural network.

Remark 3: Lemma 1 generalizes a well-known technique of computing a *symmetric* Boolean function with a depth-2 neural network [22]. In fact, Lemma 1 states that the function can be expressed *exactly* as a linear combination of the outputs from the first layer, which only takes on value 0 or 1. Thus, the "threshold power" in the second layer is not fully exploited. We make use of this observation later in reducing the depth of our division circuit.

The following lemma, using the encoding trick in [16], generalizes the result of Lemma 1 to any Boolean function that has a *constant* number of the functions in Lemma 1 as inputs.

Lemma 2: For a fixed integer $d \geq 1$, let $g(X) = h(f_1(X), \cdots, f_d(X))$, where h is an arbitrary function, and the functions f_i satisfy the assumptions stated in Lemma 1. Then $g(X)$ can be represented as a sum of polynomially many linear threshold functions in X.

Proof: By assumption, we can write each $f_i(X) = \tilde{f}_i(\sum_{j=0}^{n-1} w_{ij} x_j)$ where w_{ij} are integers bounded by a polynomial in n. Let N be the maximum of $\sum_{j=0}^{n-1} |w_{ij}|$ over all $i = 1, \cdots, d$. Now observe that for each $i = 1, \cdots, d$, the sum $\sum_{j=0}^{n-1} w_{ij} x_j$ can be determined from the value $\sum_{i=0}^{d-1} (N+1)^i (\sum_{j=0}^{n-1} w_{ij} x_j)$. But this sum is also polynomially bounded for all X. Thus, $g(X)$ is a function which depends only on a polynomially bounded weighted sum of the input variables. Our claim follows from Lemma 1. □

We shall make use of the following specific form of the above lemma. As usual, the symbol $\wedge$ denotes the logic AND function.

Corollary 1: Let $g(X) = f_1(X) \wedge f_2(X)$, where $f_1(X)$ and $f_2(X)$ are functions computed by depth-k neural networks with inputs X. Then $g(X)$ can be represented as a sum of outputs from polynomially many depth-k neural networks with inputs X.

Proof: Simply observe that each $f_i(X)$ depends on a polynomially bounded weighted sum of the outputs from depth-$(k-1)$ neural networks. Then from Lemma 2, we can write $g(X)$ as a sum of polynomially many linear threshold functions of the outputs form depth-$(k-1)$ neural networks.□

Harmonic Analysis Technique: Using the carry-look-ahead method, it is well known that the sum of two $O(n)$-bit numbers can be computed in a depth-3 circuit of polynomially many *unbounded fan-in* AND/OR gates. The depth can be reduced to 2 if threshold gates are used. Using results from harmonic analysis of Boolean functions [8], [9], it was first proved in [29] that there exists a polynomial-size depth-2 neural network for computing the sum of two $O(n)$-bit numbers. Such a depth-2 neural network was later explicitly constructed in [3]. In fact, a more general result is proved in [29]; each bit of the sum belongs to the class of Boolean functions that have *polynomially bounded spectral norms* (see [26] for definition), and every function in this class can even be *closely approximated* by a linear combination of polynomially many linear threshold functions. Since we shall make use of this result only for the sum of two $O(n)$-bit numbers, we state the result for this particular case as the following lemma.

Lemma 3: Let $X_1 = \sum_{i=0}^{n-1} x_{1_i} 2^i$ and $X_2 = \sum_{i=0}^{n-1} x_{2_i} 2^i$. Let their sum be $S = \sum_{i=0}^{n} s_i 2^i = X_1 + X_2$. For each bit s_i and any $k > 0$, there exist linear threshold functions $t_{ij}(X)$ such that

$$\left| s_i - \frac{1}{N} \sum_{j=1}^{m} w_{i_j} t_{i_j}(X) \right| \leq n^{-k}$$

where m, w_{i_j}'s, and N are integers bounded by a polynomial in n, $X = (x_{1_{n-1}}, x_{2_{n-1}}, \cdots, x_{1_0}, x_{2_0})$.

From the above lemma, each bit s_i can be expressed as $s_i(X) = \operatorname{sgn}\{(1/N)\sum_{j=1}^{m} w_{i_j} t_{i_j}(X) - 0.5\}$ and thus the sum of two $O(n)$-bit numbers can be computed in a depth-2 neural network. We would like to point out that in [3], an explicit construction of the t_{i_j}'s in the above lemma was shown.

Now, it is clear how to compute the sum of n $O(n)$-bit numbers in a depth-3 neural network. First use the "block save" technique to reduce this multiple sum to the sum of two $O(n)$-bit numbers. By Lemma 1, each bit of the resulting two numbers is a linear combination of polynomially many linear threshold functions computed in the first layer. Then take this linear combination as inputs to the depth-2 neural network for computing the sum of the resulting two numbers. Thus only three layers are needed.

To compute the product of two n-bit numbers, $x = x_{n-1}x_{n-2}\cdots x_0$, $y = y_{n-1}y_{n-2}\cdots y_0$, the first layer of our circuit outputs the n $2n$-bit numbers $z_i = z_{i_{2n-1}} z_{i_{2n-2}} \cdots z_{i_o}$ for $i = 0, \cdots, n-1$, where

$$z_i = \underbrace{0\cdots 0}_{n-i}(x_{n-1} \wedge y_i)(x_{n-2} \wedge y_i)\cdots(x_0 \wedge y_i)\underbrace{0\cdots 0}_{i}.$$

Another three layers compute the sum of all z_i's as shown before. Thus the product of 2 n-bit numbers can be computed in 4 layers.

B. Exponentiation and Modulo a "Small" Number

The Chinese Remainder Theorem can be interpreted as the fact that a positive integer $Z < P_n = \prod_{i=1}^{n} p_i$ is uniquely determined by $r_i \equiv Z \pmod{p_i}$, the values of Z modulo the primes p_i's. When we apply this result to obtain a small depth neural network for division, the primes p_i are "small" (i.e., polynomially bounded) and known in advance. We now show that an input n-bit integer modulo a "small" number can be computed in a small depth circuit.

Theorem 1: Let $X = (x_{n-1}, x_{n-2}, \cdots, x_0) \in \{0, 1\}^n$ denote an input n-bit integer, and let m be a fixed positive integer bounded by a polynomial in n. Then $X \bmod m$ can be computed in a depth-2 neural network.

Proof: Observe that if $a \equiv \tilde{a} \pmod{m}$, $b \equiv \tilde{b} \pmod{m}$, then $a + b \equiv \tilde{a} + \tilde{b} \pmod{m}$ and $a \cdot b \equiv \tilde{a} \cdot \tilde{b} \pmod{m}$. Therefore, $\sum_{i=0}^{n-1} x_i 2^i \bmod m = [\sum_{i=0}^{n-1} x_i (2^i \bmod m)] \bmod m$. Now $(2^i \bmod m) < m$ and m is polynomially bounded in n, thus each bit in the output depends only on a polynomially bounded weighted sum of the input bits. It follows from Lemma 1 that $X \bmod m$ can be computed in a depth-2 neural network. □

Before we show how division and related problems can be computed in small depth neural networks, we first show how to compute powering and exponentiation modulo a small (i.e., polynomially bounded) prime number.

Theorem 2: Let $X = (x_{n-1}, x_{n-2}, \cdots, x_0) \in \{0, 1\}^n$ be an input n-bit integer, p be a prime number bounded by a polynomial in n, and c be a positive integer not divisible by p. Then each bit of $X^n \bmod p$ and $c^X \bmod p$ can be represented by a sum of polynomially many linear threshold functions in X. Hence, $X^n \bmod p$ and $c^X \bmod p$ can both be computed in depth-2 neural networks.

Proof: Observe that $x^n \bmod p = (\sum_{i=0}^{n-1} x_i 2^i)^n \bmod p = [\sum_{i=0}^{n-1} x_i (2^i \bmod p)]^n \bmod p$. Similarly, since p is a prime, and c is not divisible by p, by Fermat's Theorem, $(c^{p-1} \equiv 1 \bmod p)$ and $(c^X \bmod p) = (c^{X \bmod (p-1)} \bmod p)$. Thus, each bit of $X^n \bmod p$ depends only on $[\sum_{i=0}^{n-1} x_i (2^i \bmod p)]$, and $c^X \bmod p$ depends only on $[\sum_{i=0}^{n-1} x_i (2^i \bmod (p-1))]$, both being a polynomially bounded weighted sum of the input bits. It follows from Lemma 1 that each bit of $X^n \bmod p$ and $c^X \bmod p$ can be represented as a sum of polynomially many linear threshold functions in X, and hence can be computed in a depth-2 neural network. □

Remark 4: First notice that if c is divisible by p, then trivially $c^X \equiv 0 \bmod p$. Also, it is important to note that in the above theorem, the numbers p and c are known in advance. The only input variables are the $X = (x_{n-1}, x_{n-2}, \cdots, x_0)$. We made crucial use of this fact in the proof of Theorem 2. Further, notice that $c^X \bmod p$ can be represented in $O(\log$

n) bits, whereas c^X requires exponentially many bits (since X is exponentially large). We restrict ourselves in the EXPONENTIATION function c^Y to input Y that is polynomially bounded in n.

Theorem 3: EXPONENTIATION can be computed in a depth-2 neural network.

Proof: Let Y be an input positive number $\leq n^k$, and c be a fixed positive integer. Since each bit of the $O(n^k)$-bit representation of c^Y only depends on Y which is polynomial bounded, it follows from Lemma 1 that c^Y can be computed in a depth-2 neural network. □

C. Powering in Depth-4

Recall that multiplication of two n-bit numbers can be computed in a depth-4 neural network. We showed this result by reducing multiplication to the sum of n $O(n)$-bit numbers. It is not clear if the function POWERING can be similarly reduced to a multiple sum of polynomially many $O(n^k)$-bit numbers. Intuitively, this function seems much "harder" in terms of circuit complexity than multiplication. Using the techniques we developed in previous sections and the Chinese Remainder Theorem, we now show that, as in the case of multiplication, POWERING only requires at most 4 layers of threshold gates to compute.

Theorem 4: POWERING can be computed in a depth-4 neural network.

Note that for any *fixed* c, this result also implies that we can compute X^{n^c} in a depth-4 neural network (with polynomial size).

Before we prove the above theorem, we need to introduce some more techniques to reduce the depth of the network. Suppose $f(X) = \operatorname{sgn}\{\sum_{i=1}^m w_i t_i(X)\}$. If each $t_i(X)$ can be computed in a depth-k neural network, then clearly $f(X)$ can be computed in a depth-$(k+1)$ neural network. The following lemma states that we can reduce the depth by one if each $t_i(X)$ can be *closely approximated* by a linear combination of outputs from polynomially many depth-$(k-1)$ neural networks.

Lemma 4: Suppose $f(X) = \operatorname{sgn}\{\sum_{i=1}^m w_i t_i(X)\}$, where $\sum_{i=1}^m w_i t_i(X) \neq 0$ for all inputs X, and $\sum_{i=1}^m |w_i| \leq n^c$. If, for each i,

$$\left| t_i(X) - \frac{1}{N}\sum_{j=1}^{m_i} \tilde{w}_{i_j}\tilde{t}_{i_j}(X) \right| < n^{-c}$$

where m_i's, $\tilde{w}_{i_j}$'s, and N are integers bounded by a polynomial in n, and each $\tilde{t}_{i_j}$ can be computed in a depth-$(k-1)$ neural network, then $f(X)$ can be computed in a depth-k neural network.

Proof: Let $r_i(X) = (1/N)\sum_{j=1}^{m_i} \tilde{w}_{i_j}\tilde{t}_{i_j}(X)$. By assumption, $t_i(X) = r_i(X) + \epsilon_i(X)$ where $|\epsilon_i(X)| < n^{-c}$. Then

$$\operatorname{sgn}\left\{\sum_{i=1}^m w_i r_i(X)\right\} = \operatorname{sgn}\left\{\sum_{i=1}^m w_i (t_i(X) - \epsilon_i(X))\right\}$$

$$= \operatorname{sgn}\left\{\sum_{i=1}^m w_i t_i(X) - \epsilon\right\}$$

where $\epsilon = \sum_{i=1}^m w_i \epsilon_i$.

Note that

$$|\epsilon| = \left|\sum_{i=1}^m w_i \epsilon_i\right| < n^{-c}\sum_{i=1}^m |w_i| \leq 1.$$

Since, by assumption, $\sum_{i=1}^m w_i t_i(X)$ is a nonzero integer for all X, therefore,

$$f(X) = \operatorname{sgn}\left\{\sum_{i=1}^m w_i t_i(X)\right\} = \operatorname{sgn}\left\{\sum_{i=1}^m w_i r_i(X)\right\}$$

$$= \operatorname{sgn}\left\{\sum_{i=1}^m w_i \left(\frac{1}{N}\sum_{j=1}^{m_i} \tilde{w}_{i_j}\tilde{t}_{i_j}(X)\right)\right\}.$$

By assumption, each $\tilde{t}_{i_j}(X)$ can be computed in a depth-$(k-1)$ neural network. Hence, $f(X) = \operatorname{sgn}\{\sum_{i=1}^m w_i(\sum_{j=1}^{m_i} \tilde{w}_{ij}\tilde{t}_{i_j}(X))\}$ can be computed in a depth-k neural network. □

Remark 5: Lemma 3 of Section III-A states that each bit of the sum of two n-bit numbers can be *closely approximated* by a linear combination of outputs from polynomially many linear threshold gates. It follows from our reduction technique that each bit of the sum of n $O(n)$-bit numbers can be closely approximated by a linear combination of outputs from polynomially many depth-2 neural networks.

We need a slight generalization of Lemma 4 in our proof of Theorem 4. The following lemma states that if t_1 and t_2 can be closely approximated by a polynomially bounded sum of outputs from depth-k neural networks, so can their product $t_1 \wedge t_2$.

Lemma 5: Suppose for $i = 1, 2$, and for every $c > 0$, there exist integers m_i, w_i, and N that are bounded by a polynomial in n such that for all inputs X

$$\left| t_i(X) - \frac{1}{N}\sum_{j=1}^{m_i} w_{i_j} t_{i_j}(X) \right| = O(n^{-c})$$

where each t_{i_j} can be computed in a depth-k neural network. Then, there exist integers $\tilde{m}$, $\tilde{w}_j$, and $\tilde{N}$ that are bounded by a polynomial in n such that

$$\left| t_1(X) \wedge t_2(X) - \frac{1}{\tilde{N}}\sum_{j=1}^{\tilde{m}} \tilde{w}_j \tilde{t}_{i_j}(X) \right| = O(n^{-c})$$

where each $\tilde{t}_j$ can be computed in a depth-k neural network.

Proof: Let $c > 0$ be given. For $i = 1, 2$, let $t_i(X) = (1/N)\sum_{j=1}^{m_i} w_{i_j} t_{i_j}(X) + \epsilon_i(X)$, where $|\epsilon_i(X)| = O(n^{-c})$ by assumption. Since each $t_i(X)$ is either 0 or 1, thus $|(1/N)\sum_{j=1}^{m_i} w_{i_j} t_{i_j}(X)| \leq 1 + O(n^{-c})$. Moreover,

$$t_1(X) \wedge t_2(X) = t_1(X)t_2(X)$$

$$= \frac{1}{N^2}\sum_{j=1}^{m_1}\sum_{l=1}^{m_2} w_{1_j} w_{2_l} t_{1_j}(X) t_{2_l}(X) + O(n^{-c}).$$

Now, by Corollary 1, each $t_{1_j}(X)t_{2_l}(X) = t_{1_j}(X) \wedge t_{2_l}(X)$ can be represented as a sum of outputs from polynomially many depth-k neural networks. The conclusion follows by substituting this sum into the previous expression for $t_1(X) \wedge t_2(X)$. □

Remark 6: Clearly, we can apply Lemma 5 repeatedly so that the result can be generalized to the product of a constant number of $t_i(X)$'s. In other words, the product of a constant number of $t_i(X)$'s can also be closely approximated by a polynomially bounded sum of outputs from depth-k neural networks.

Recall that each threshold gate in our circuit is only allowed to have polynomially bounded integer weights. The following lemma states that a linear threshold gate with arbitrary (even real-value) weights can be *closely approximated* by a linear combination of outputs from polynomially many depth-2 neural networks. The proof can be found in [29].

Lemma 6: Let $f(X)$ be a linear threshold function (of n variables) whose weights can be arbitrary real numbers. Then for any $k > 0$, there exists a linear combination of functions $t_j(X)$ computable in depth-2 neural networks (with polynomially bounded integer weights)

$$F(X) = \frac{1}{n}\sum_{j=1}^{s} w_j t_j(X)$$

such that

$$|F(X) - f(X)| \leq n^{-k},$$

where s, w_j's, and N are integers bounded by a polynomial in n.

With the above lemmas, we are ready to prove Theorem 4.

Proof of Theorem 4: Let X denote an input n-bit integer. We want to compute the n^2-bit representation of $Z = X^n$. Our main tool will be the Chinese Remainder Theorem. This number theoretic result was used in [6] to construct $O(\log n)$-depth and polynomial size AND/OR circuits for division. It allows us to first compute Z modulo small prime numbers in parallel and then combine the results using small depth neural networks.

Let p_i denote the ith prime number, and let $\pi(k)$ denote the number of primes $\leq k$. Let

$$P_n = \prod_{i=1}^{\pi(n^2)} p_i$$

be the product of all primes $\leq n^2$. Let

$$q_i = P_n/p_i, \quad \tilde{q}_i = q_i^{-1} \bmod p_i, \quad \text{and} \quad m_i = q_i \cdot \tilde{q}_i;$$

$\tilde{q}_i$ exists since p_i is a prime. Observe that by the *Prime Number Theorem*, P_n is greater than 2^{n^2} for sufficiently large n. Moreover, $(Z \bmod P_n) = Z$ since $Z < 2^{n^2} < P_n$. We shall compute $(Z \bmod P_n)$ with a small depth threshold circuit via the Chinese Remainder Theorem as follows:

1) for $i = 1, \cdots, n^2$, compute in parallel the values $r_i = Z \bmod p_i$;
2) $\tilde{Z} = \sum_{i=1}^{\pi(n^2)} r_i \cdot m_i$;
3) $Z = (Z \bmod P_n) = (\tilde{Z} \bmod P_n)$.

It is important to note that the m_i are known in advance, and thus step 2 above is in fact multiple addition. Also note that $r_i \leq n^2$ and $m_i \leq P_n$, therefore, $\tilde{Z} \leq \sum_{i=1}^{\pi(n^2)} n^2 P_n \leq n^4 \cdot P_n$. Hence, $Z = (\tilde{Z} \bmod P_n) = \tilde{Z} - k \cdot P_n$ for some k, where $0 \leq k \leq n^4$. For each $k \in \{0, \cdots, n^4\}$, let

$$\begin{aligned} EQ_k(Z) &= \operatorname{sgn}\{\tilde{Z} - k \cdot P_n\} \\ &\quad + \operatorname{sgn}\{(k+1)P_n - \tilde{Z} - 1\} - 1 \\ &= \begin{cases} 1, & \text{if } Z = (\tilde{Z} \bmod P_n) = \tilde{Z} - k \cdot P_n, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Let z_{jk} be the jth bit of $\tilde{Z} - k \cdot p_n$. Then the jth bit of Z is

$$\bigvee_{0 \leq k \leq n^4} (EQ_k(Z) \wedge z_{jk}).$$

Now let us calculate the number of layers needed to compute each jth bit of Z. We can compute the values r_i in Step 1 as a sum of the outputs from polynomially many linear threshold gates in the first layer by Theorem 2. By expressing $(-k \cdot P_n)$ in 2's complement representation, $(\tilde{Z} - k \cdot P_n)$ is simply a multiple sum (with input variables r_i's). It follows from Remark 5 that each z_{jk} can be *closely approximated* as a sum of outputs from polynomially many depth-2 neural networks whose inputs are the variables r_i. Similarly, Lemma 6 implies that the functions $EQ_k(Z)$ can also be *closely approximated* as a sum of outputs from polynomially many depth-2 neural networks whose inputs are the variables r_i. Thus, $EQ_k(Z)$ and z_{jk} can be *closely approximated* as a sum of the outputs from polynomially many depth-3 neural networks whose inputs are the variables X. Now Lemma 5 implies that $(EQ_k(Z) \wedge z_{jk})$ can also be closely approximated as a sum of the outputs from polynomially many depth-3 neural networks. Then, it follows from Lemma 4 that the output $\vee_{0 \leq k \leq n^4}(EQ_k(Z) \wedge z_{jk})$ can be computed in a depth-4 neural network. □

D. Multiple Product in Depth-5

Given n n-bit numbers z_i, $i = 1, \cdots, n$, we want to compute the n^2-bit representation of (MULTIPLE PRODUCT). We are going to show that this function requires at most 5 layers of threshold gates to compute.

Theorem 5: MULTIPLE PRODUCT can be computed in a depth-5 neural network.

Proof: We use the same notation as in the proof of Theorem 4. Let $Z = \prod_{j=1}^{n} z_j$. Again, the proof is similar to the proof of Theorem 4, except that it is not obvious how to compute each $r_i = Z \bmod p_i$. In the case of POWERING, each $r_i = (c^Y \bmod p_i)$ or $(X^n \bmod p_i)$ can be expressed as a sum of polynomially many linear threshold functions. To prove Theorem 5, it suffices to show that in the case of MULTIPLE PRODUCT, each $r_i = \prod_{j=1}^{m} z_j \bmod p_i$ can be expressed as a sum of the outputs from polynomially many depth-2 neural networks. It then follows that we can compute MULTIPLE PRODUCT with a depth-5 neural network.

Let Z_{p_i} be the set of all integers modulo p_i. Since each p_i is a prime number, Z_{p_i} is a finite field. Let g_i be a

generator of Z_{p_i}, i.e., every integer $\tilde{Z} \equiv g_i^{\tilde{e}_i} \pmod{p_i}$ for some $\tilde{e}_i \in \{0, \cdots, p_i - 1\}$. Note that $r_i = (\prod_{j=1}^{n} z_j \bmod p_i) = \prod_{j=1}^{n} (z_j \bmod p_i) \bmod p_i$. The idea is to compute $e_{i,j}$, the exponent of $(z_j \bmod p_i)$ with respect to the generator g_i, i.e., $(z_j \bmod p_i) = (g_i^{e_{i,j}} \bmod p_i)$. Then,

$$r_i = \prod_{j=1}^{n} z_j \bmod p_i = \prod_{j=1}^{n} g_i^{e_{i,j}} \bmod p_i = g_i^{\left(\sum_{j=1}^{n} e_{i,j}\right)} \bmod p_i.$$

Now, observe that each bit of $e_{i,j}$ is a function of $(z_j \bmod p_i)$, which is a polynomially bounded weighted sum of the bits in z_j. By Lemma 1, $e_{i,j}$ can be represented as a sum of polynomially many linear threshold functions in the input bits of z_j. Since each $e_{i,j} < p_i, \sum_{j=1}^{n} e_{i,j} < np_i$ which is polynomially bounded. Moreover, each r_i only depends on $\sum_{j=1}^{n} e_{i,j}$. Again, Lemma 1 implies that r_i can be represented as a sum of polynomially many linear threshold functions in the input bits of $e_{i,j}$. It follows that r_i can be expressed as a sum of the outputs from polynomially many depth-2 neural networks, with inputs z_j. □

E. Division in Depth 4

The last main result in this paper is to show that division can be computed in a small depth neural network. In particular, we shall see that $\mathrm{DIV}_k(x/y)$ can be computed in a depth-4 neural network. First we need a lemma that is a slight generalization of the result of Theorem 2.

Lemma 7: Let N, p_i, m_i, a_i, b_i, c_i, K be fixed integers, where N and p_i are positive integers bounded by a polynomial in n, and a_i, b_i, and c_i are integers with a polynomially bounded number of bits. Let x and y be the input integers with a polynomially bounded number of bits. Let $Z = \sum_{j=1}^{N} a_j x (b_j + c_j y)^j$. Let $r_i \equiv Z \pmod{p_i}$, where r_i is polynomially bounded. Then, each bit $\tilde{z}_l$ of

$$\tilde{Z} = \sum_{i=1}^{N} r_i \cdot m_i - K$$

can be closely approximated by a linear combination of outputs from depth-3 neural networks, i.e., for any $k > 0$, there exists a linear combination of functions t_j computable in depth-3 neural networks (with polynomially bounded integer weights) such that

$$\left| \tilde{z}_l - \frac{1}{M} \sum_{j=1}^{s} w_j t_j \right| \leq n^{-k}$$

where s, w_j's, and M are integers bounded by a polynomial in n.

Proof: We first show that each term $z_{i,j} = a_j x (b_j + c_j y)^j \bmod p_i$ can be represented as a sum of polynomially many linear threshold functions with inputs x and y. Note that $z_{i,j} = \{a_j (x \bmod p_i)[(b_j + c_j y) \bmod p_i]^j\} \bmod p_i$. Let $x \equiv \sum_l \hat{w}_l x_l \pmod{p_i}$ and $(b_j + c_j y) \equiv \sum_l \tilde{w}_l y_l \pmod{p_i}$ (the weights are polynomially bounded). Let $M_y = \sum_l |\tilde{w}_l| + 1$. Then, each bit of $z_{i,j}$ can be expressed as a function of $(M_y \sum_l \hat{w}_l x_l) + (\sum_l \tilde{w}_l y_l)$, which is still a polynomially bounded weighted sum of the input bits of x and y. By Lemma 1, each bit of $z_{i,j}$ can be represented as a sum of polynomially many linear threshold functions with input bits x and y.

Now, $\tilde{Z} = \sum_{i=1}^{N} (\sum_{j=1}^{N} z_{i,j}) \cdot m_i - K$, which is simply a multiple sum, since m_i's are known in advance. From Remark 5 of Section III-C, each bit of this sum can be closely approximated by a linear combination of outputs from polynomially many depth-2 neural networks with inputs $z_{i,j}$. Since each bit of $z_{i,j}$ can be represented as a sum of polynomial many linear threshold functions with input bits x and y, the lemma follows. □

Theorem 6: Let x and $y > 0$ be two n-bit integers. Then $\mathrm{DIV}_k(x/y)$ can be computed in a depth-4 neural network.

Proof: Note that $\mathrm{DIV}_k(x/y) = 2^{-k}\, \mathrm{DIV}_0(2^k x/y)$, so it suffices to prove our claim for the case $k = 0$. The resulting neural networks for the general case when k is polynomial in n have the same depth and the size will increase by a polynomial factor.

The underlying idea is to compute an *over-approximation* $\tilde{a}$ to x/y such that $x/y \leq \tilde{a} \leq x/y + 2^{-(n+1)}$. We claim that $\lfloor \tilde{a} \rfloor = \lfloor x/y \rfloor$. Clearly, the claim is true if x/y is an integer. Suppose x/y is not an integer. Then,

$$x = \lfloor x/y \rfloor y + r,$$

where $0 < r < y < 2^n$. Thus,

$$x/y - \lfloor x/y \rfloor = r/y \geq 2^{-n},$$

and

$$\lfloor x/y \rfloor + 1 - x/y = 1 - r/y \geq 2^{-n}.$$

In other words,

$$\lfloor x/y \rfloor + 2^{-n} \leq x/y \leq \lfloor x/y \rfloor + 1 - 2^{-n}.$$

Since $0 \leq \tilde{a} - x/y \leq 2^{(2n+1)}$ by assumption, we must have

$$\lfloor x/y \rfloor < \tilde{a} < \lfloor x/y \rfloor + 1.$$

Hence, $\lfloor \tilde{a} \rfloor = \lfloor x/y \rfloor$.

Since x/y is equal to the product of x and y^{-1}, it is enough to get an over-approximation $\tilde{y}^{-1}$ of y^{-1} with error $\leq 2^{-(2n+1)}$. Then, using Lemma 7, we can compute $q = x \cdot \tilde{y}^{-1}$ with error $< 1/2$ with a small depth neural network.

To construct an over-approximation of y^{-1}, let $j \geq 1$ be the integer such that $2^{j-1} \leq y < 2^j$. Note that $|1 - y2^{-1}| \leq 1/2$, and we can express y^{-1} as a series expansion

$$y^{-1} = 2^{-j} \cdot (1 - (1 - y2^{-j}))^{-1} = 2^{-j} \sum_{i=0}^{\infty} (1 - y2^{-j})^i.$$

If we put

$$\tilde{y}^{-1} = 2^{-j} \sum_{i=0}^{2n} (1 - y2^{-j})^i,$$

then

$$0 \leq (y^{-1} - \tilde{y}^{-1}) \leq 2^{-j} \sum_{i=(2n+1)}^{\infty} 2^{-i} \leq 2^{-(n+1)}.$$

Since $x < 2^n$, we have

$$0 \leq (xy^{-1} - x\tilde{y}^{-1}) < 2^{-(n+1)}.$$

Suppose for the moment that we can find the integer $j \geq 1$ such that $2^{j-1} \leq y < 2^j$. Now, we can rewrite

$$x\tilde{y}^{-1} = \frac{1}{2^{j(2n+2)}} \sum_{i=0}^{2n+1} 2^{j(2n+1-i)} x(2^j - y)^i.$$

Using Lemma 7, we can proceed as in the proof of Theorem 4 to construct a small depth neural network for $x\tilde{y}^{-1}$.

Let $Z_j = \sum_{i=0}^{2n+1} 2^{j(2n+1-i)} x(2^j - y)^i$. Then, $x\tilde{y}^{-1} = \frac{1}{2^{j(2n+2)}} Z_j$, a shifting of the bits in Z_j. Let N be a sufficiently large integer such that the product of the first N primes $\prod_{i=1}^{N} p_i = P_N > Z_j$ for all $j = 1, \cdots, n$. It is easy to see that all p_i's are polynomially bounded (in n). Let

$$q_i = P_N/p_i, \qquad \tilde{q}_i = q_i^{-1} \bmod p_i, \qquad \text{and} \quad m_i = q_i \cdot \tilde{q}_i.$$

Again, we compute Z_j via the Chinese Remainder Theorem as follows.

1) For $i = 1, \cdots, N$, compute in parallel the values $r_{i,j} = Z_j \bmod p_i$;
2) $\tilde{Z}_j = \sum_{i=1}^{N} r_{i,j} \cdot m_i$;
3) $Z_j = (Z_j \bmod P_N) = (\tilde{Z}_j \bmod P_N)$.

Note that $\tilde{Z}_j \leq n^\alpha P_N$ for some $\alpha > 0$. Hence, $Z_j = (\tilde{Z}_j \bmod P_N) = \tilde{Z}_j - kP_n$ for some k, where $0 \leq k \leq n^\alpha$. For each $k \in \{0, \cdots, n^\alpha\}$, let

$$\begin{aligned} EQ_k(Z_j) &= \operatorname{sgn}\{\tilde{Z}_j - kP_N\} \\ &\quad + \operatorname{sgn}\{(k+1)P_N - \tilde{Z}_j - 1\} - 1 \\ &= \begin{cases} 1, & \text{if } Z_j = (\tilde{Z}_j \bmod P_N) = \tilde{Z}_j - kP_N, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Let $\sum_l z_{j,k,l} 2^l = \frac{1}{2^{j(2n+2)}}(\tilde{Z}_j - kP_N)$. If $EQ_{k^*}(Z_j) = 1$, i.e., $(\tilde{Z}_j \bmod P_N) = \tilde{Z}_j - k^* P_N$, then

$$\operatorname{DIV}_0(x/y) = \sum_{l=0}^{n-1} z_{j,k^*,l} 2^l.$$

Thus, the ith bit of $\operatorname{DIV}_0(x/y)$ can be computed as

$$\bigvee_{1 \leq k \leq n^\alpha} (EQ_k(Z_j) \wedge z_{j,k,i}.$$

This expression is based on the assumption that we can find the unique integer $j \geq 1$ such that $2^{j-1} \leq y < 2^j$. We can compute such integer j in parallel without increasing the depth of the circuit. To see this, for each $j \in \{1, \cdots, n\}$, let

$$\begin{aligned} I_j &= \operatorname{sgn}\{y - 2^{j-1}\} + \operatorname{sgn}\{2^j - y - 1\} - 1 \\ &= \begin{cases} 1, & \text{if } 2^{j-1} \leq y < 2^j, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Then, the ith bit of $\operatorname{DIV}_0(x/y)$ is

$$\bigvee_{1 \leq j \leq n} \bigvee_{0 \leq k \leq n^\alpha} (I_j \wedge EQ_k(Z_j) \wedge z_{j,k,i}).$$

Note that (by Lemma 7) each of the $z_{j,k,i}$, $EQ_k(Z_j)$, and I_j can be closely approximated by a linear combination of outputs from polynomially many depth-3 neural networks, with inputs x and y. Then, Remark 6 following Lemma 5 implies that each $(I_j \wedge EQ_k(Z_j) \wedge z_{j,k,i})$ can also be closely approximated by a linear combination of outputs from polynomially many depth-3 neural networks. Hence, it follows from Lemma 4 that the final result can be computed in a depth-4 neural network. □

F. Sorting in Depth-3

It was shown in [29] that the SORTING of n n-bit numbers can be computed in a depth-4 neural network. By using Corollary 1, we can improve the depth of the sorting network presented in [29]. Furthermore, we also prove that any polynomial-size neural network for SORTING must have a depth of at least 3. Thus, our neural network for SORTING is optimal in depth.

In SORTING, we assume that the input is a list of n n-bit binary numbers and the output will be the same list sorted in nondecreasing order. A number which appears m times in the input list will be duplicated m times in the output list.

Theorem 7: The SORTING of n n-bit numbers can be computed in a depth-3 neural network.

Proof: Let $z_i = z_{i_n} z_{i_{n-1}} \cdots z_{i_1}$, for $i = 1, \cdots, n$, denote the input binary numbers. Define

$$c_{ij} = \begin{cases} 1, & \text{if } z_i > z_j \quad \text{or } (z_i = z_j \quad \text{and} \quad i \geq j), \\ 0, & \text{otherwise.} \end{cases}$$

If we let $p_i = \sum_{j=1}^{n} c_{ij}$, then p_i is the position of z_i in the sorted list. Let $L_m(p_i) = \operatorname{sgn}\{p_i - m\}$; then $L_m(p_i) = 1$ if $p_i \geq m$ and 0 otherwise. Similarly, let $l_m(p_i) = \operatorname{sgn}\{m - p_i\}$; then $l_m(p_i) = 1$ if $p_i \leq m$ and 0 otherwise, Thus, the kth bit of the mth number in the sorted list is

$$\bigvee_{1 \leq i \leq n} (L_m(p_i) \wedge l_m(p_i) \wedge z_{ik}),$$

where $\vee$ and $\wedge$, respectively, denote the OR and AND functions.

In [29], it was shown that each c_{ij} can be closely approximated by a sum of polynomially many linear threshold functions. Hence, by Lemma 4, each $L_m(p_i)$ and $l_m(p_i)$ can be computed in a depth-2 neural network. Now apply Corollary 1 twice; it follows that each $L_m(p_i) \wedge l_m(p_i) \wedge z_{ik}$ can be represented as a sum of outputs from polynomially many depth-2 neural networks. We need one more threshold gate to compute the output $\bigvee$ gate. Thus, altogether, only 3 layers are needed. □

In [14], it was shown that any polynomial size neural network computing the function Inner Product Modulo 2, i.e., $IP(x_1, \cdots, x_n, y_1, \cdots, y_n) := x_1 y_1 \oplus \cdots \oplus x_n y_n$ must have depth at least 3. We shall provide a similar lower bound result for SORTING by reducing it to Inner Product Modulo 2.

Theorem 8: Any polynomial size neural network for SORTING must have depth at least 3.

Proof: We show that if we can sort $2n + 1$ integers of length (log n + 3)-bits, then the Inner Product Modulo 2 function $IP(x_1, \cdots, x_n\, y_1, \cdots, y_n)$ can be computed by setting some of the inputs of SORTING to variables x_i, y_i, or constants. Let $b(i)$ be the log n-bit binary representation of the integer i, for $i = 0, \cdots, n-1$. We choose the following input to sorting (each line represents one integer):

$$\begin{aligned} &x_1 y_1 1 \cdots 1 \\ &x_2 y_2 1 \cdots 1 \\ &\cdots \\ &x_n y_n 1 \cdots 1 \end{aligned}$$

$110b(0)$
$110b(1)$
$\cdots$
$110b(n-1)$
$110b(n)$.

Note that each of the above $2n+1$ integers has the same number of bits in its binary representation. We claim that the least significant bit of the $(n+1)^{st}$ integer in the output list (in ascending order) always yields the value of the $IP(x_1, \cdots, x_n, y_1, \cdots, y_n)$ function.

To justify our claim, consider the position in the output list represented by the integer $110b(k)$. Clearly, $110b(j)>110b(k)$ if $j > k$, and $x_i y_i 1 \cdots 1 > 110b(k)$, if and only if $(x_i, y_i) = (1, 1)$. Hence, if $x_i y_i = 1$ for exactly r many i's, then there are exactly $n-k+r$ many integers which are greater than 110b(k). Moreover, $IP(x_1, \cdots, x_n, y_1, \cdots, y_n)$ is 1 if r is odd and 0 otherwise, and is the same as the least significant bit of $b(r)$.

Choosing $k = r$ reveals that the integer which appears at position $n+1$ in the sorting list is the integer 110b(r). The least significant bit of this integer is identical with the output of $IP(x_1, \cdots, x_n, y_1, \cdots, y_n)$. Hence, a network for SORTING can be used for computing the Inner Product Modulo 2 function. By the lower bound result in [14], we conclude that any polynomial size neural network for SORTING has depth at least 3. □

IV. Conclusion

In this paper, we have shown the construction of small depth neural networks for division and relation problems. In particular, we prove that powering and division are computable in polynomial-size depth-4 neural networks, multiple product is computable in polynomial-size depth-5 neural networks, and sorting is computable in a polynomial-size depth-3 neural network. We would like to point out a recent improvement [30] of the results in this paper. It was shown in [30] that in computing multiple sum, each bit can be closely approximated by the outputs of polynomially many linear threshold functions. It follows from this result that the depth of our networks for division, powering, and multiple product can all be reduced by 1. Moreover, multiplication can be computed in depth-3 polynomial-size neural networks. The networks for multiplication and division constructed in [30] are optimal in depth. The results in [30] also employ techniques from [12], [13]. Details can be found in [30].

We have focused our study on the depth of neural networks, while the only restriction on the size is that it be bounded by a polynomial in the number of inputs. As a result, the upper bound on the size of our network is possibly far from being optimal. A natural continuation of our work is to try to reduce the size of our network without increasing the depth. It will also be interesting to prove lower bound results on the depth of neural networks even if the size is restricted.

References

[1] Y. S. Abu-Mostafa and D. Psaltis, "Optical neural computers," *Sci. Amer.*, vol. 256, no 3, pp. 88–95, 1987.

[2] M. Ajtai, $\sum_1^1$-formulae on finite structures," *Ann. Pure Appl. Logic*, vol. 24, pp. 1–48, 1983.

[3] N. Alon and J. Bruck, "Explicit constructions of depth-2 majority circuits for comparison and addition," IBM Res. Rep., RJ 8300, Aug. 1991.

[4] L. Atlas *et al.*, "A performance comparison of trained multilayer perceptrons and trained classification trees," *Proc. IEEE*, vol. 78, pp. 1614–1619, 1990.

[5] E. B. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity*, vol. 4, pp. 193–215, 1988.

[6] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," *Siam J. Comput.*, vol. 15, pp. 994–1003, 1986.

[7] R. Boppana and M. Sipser, "The complexity of finite functions," in *Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity*, J. Van Leeuwen, Ed. Cambridge, MA: MIT Press, 1990.

[8] J. Bruck, "Harmonic analysis of polynomial threshold functions," *SIAM Discrete Math.*, pp. 168–177, May 1990.

[9] J. Bruck and R. Smolensky, "Polynomial threshold functions, AC^0 functions and spectral norms," presented at the IEEE Symp. Found. Comput. Sci., Oct. 1990.

[10] A. K. Chandra, L. Stockmeyer, and U. Vishkin, "Constant depth reducibility," *Siam J. Comput.*, vol. 13, pp. 423–439, 1984.

[11] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits and the polynomial-time hierarchy," in *Proc. IEEE Symp. Found. Comput. Sci.*, vol. 22, pp. 260–270, 1981.

[12] M. Goldmann, J. Håstad, and A. Razborov, "Majority gates vs. general weighted threshold gates," presented at the *Seventh Ann. Conf. on Structure in Complexity Theory.*

[13] M. Goldmann and M. Karpinski, "Constructing depth $d+1$ majority circuits that simulate depth d threshold circuits," unpublished manuscript.

[14] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan, "Threshold circuits of bounded depth," in *Proc. IEEE Symp. Found. Comput. Sci.*, vol. 28, pp. 99–110, 1987.

[15] J. Håstad, "Almost optimal lower bounds for small depth circuits," *ACM Symp. Theor. Comput.*, vol. 18, pp. 6–20, 1986.

[16] T. Hofmeister, W. Hohberg, and S. Köhling, "Some notes on threshold circuits and multiplication in depth 4," *Inform. Processing Lett.*, vol. 39, pp. 219–225, 1991.

[17] T. Hofmeister and P. Pudlák, "A proof that division is not in TC_2^0," Forschungsbericht Nr. 447, Uni Dortmund, 1992.

[18] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, vol. 79, pp. 2554–2558, 1982.

[19] T. Kohonen, *Self-Organization and Associative Memory.* Berlin, Heidelberg, Germany: Springer-Verlag, 1989.

[20] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1.* Cambridge, MA: MIT Press, 1986.

[21] W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bull. Math. Biophy.*, vol. 5, pp. 115–133, 1943.

[22] S. Muroga, "The principle of majority decision logic elements and the complexity of their circuits, presented at the Int. Conf. Inform. Processing, Paris, France, June 1959.

[23] ——, *Threshold Logic and Its Applications.* New York: Wiley, 1971.

[24] I. Niven, H. Zuckerman, and H. Montgomery, *An Introduction to the Theory of Numbers*, 5th ed. New York: Wiley, 1991.

[25] I. Parberry and G. Schnitger, "Parallel computation with threshold functions," *J. Comput. Syst. Sci.*, vol. 36, no. 3, pp. 278–302, 1988.

[26] J. Reif, "On threshold circuits and polynomial computation," in *Proc. Structure Complexity Theory Symp.*, 1987, pp. 118–123.

[27] J. Reif and S. Tate, "On threshold circuits and efficient, constant depth polynomial computations," Aug. 1988, accepted for publication in *SICOMP.*

[28] K.-Y. Siu and J. Bruck, "Neural computation of arithmetic functions," *Proc. IEEE*, vol. 78, pp. 1669–1675, Oct. 1990 (Special Issue on Neural Networks).

[29] ——, "On the power of threshold circuits with small weights," *SIAM J. Discrete Math.*, vol. 4, no. 3, pp. 423–435, Aug. 1991.

[30] K.-Y. Siu and V. Roychowdhury, "On optimal depth threshold circuits for multiplication and related problems," to appear in *SIAM J. Discrete Math.*

[31] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, 1964.

[32] I. Wegener, *The Complexity of Boolean Functions.* New York: Wiley, 1987.

[33] A. Yao, "Separating the polynomial-time hierarchy by oracles," in *Proc. IEEE Symp. Found. Comput. Sci.*, 1985, pp. 1–10.